Introduction to Parallelism & Parallelism on HPC

Examples in Julia and Python

Criston Hyett

December 7, 2022

What is parallelism?

- The common idea of **divide and conquer**
- Works extremely well in many relevant scenarios
  - Large dimension linear algebra, (i.e. ML)
  - Monte Carlo
  - Ordinary/Partial/Stochastic Differential equations (big linear algebra + Monte Carlo)
- Not a panacea!
  - Inefficient code can be inefficient *and* consume plenty of cpu-hours on the HPC

When should you care?

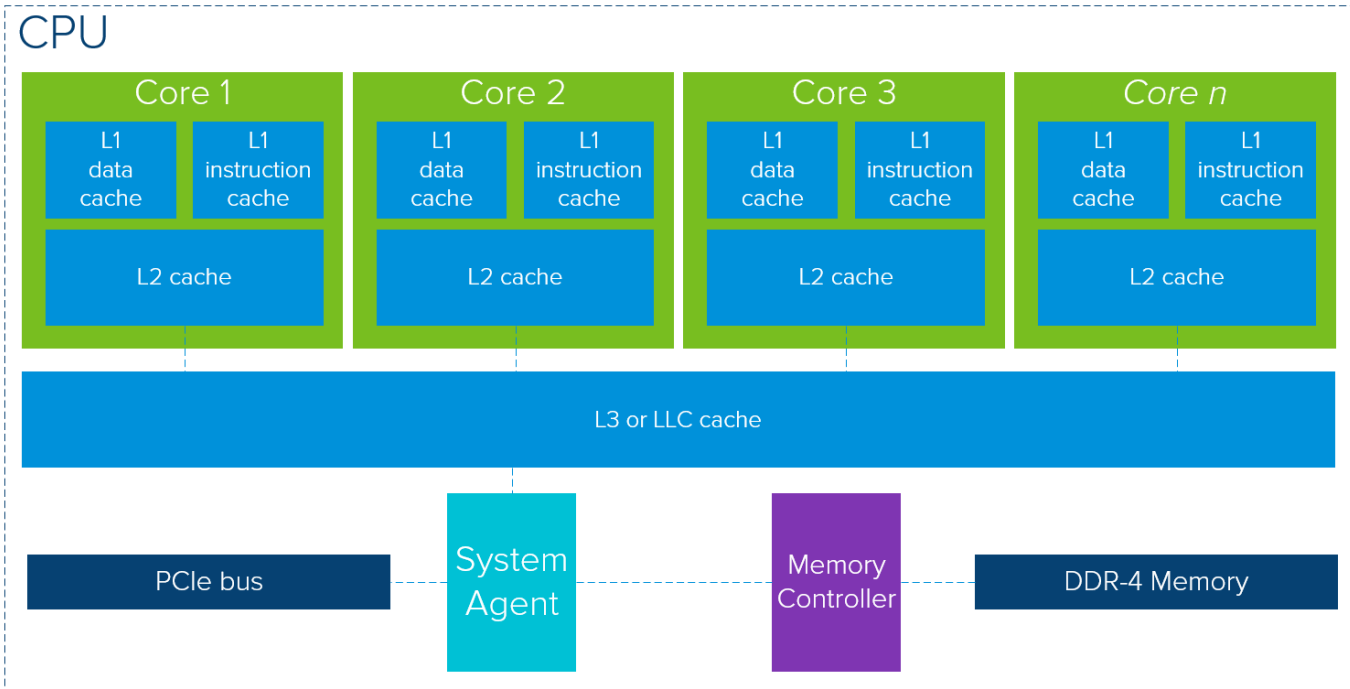
- When the benefits outweigh the costs!
- Costs
  - **Human costs:** Coding, debugging, refactoring
  - **Computational Costs:** parallelism requires coordination between threads, and/or nodes. When this coordination is required often, or significant data is passing between threads, parallel benefits may be outweighed.
- Benefits
  - **Computation speed**
  - **Smaller memory overheads per thread**
  - Process independence

When should you care?

- When the benefits outweigh the costs!
- Costs
  - **Human costs:** Coding, debugging, refactoring
  - **Computational Costs:** parallelism requires coordination between threads, and/or nodes. When this coordination is required often, or significant data is passing between threads, parallel benefits may be outweighed.
- Benefits
  - **Computation speed**
  - **Smaller memory overheads per thread**
  - Process independence

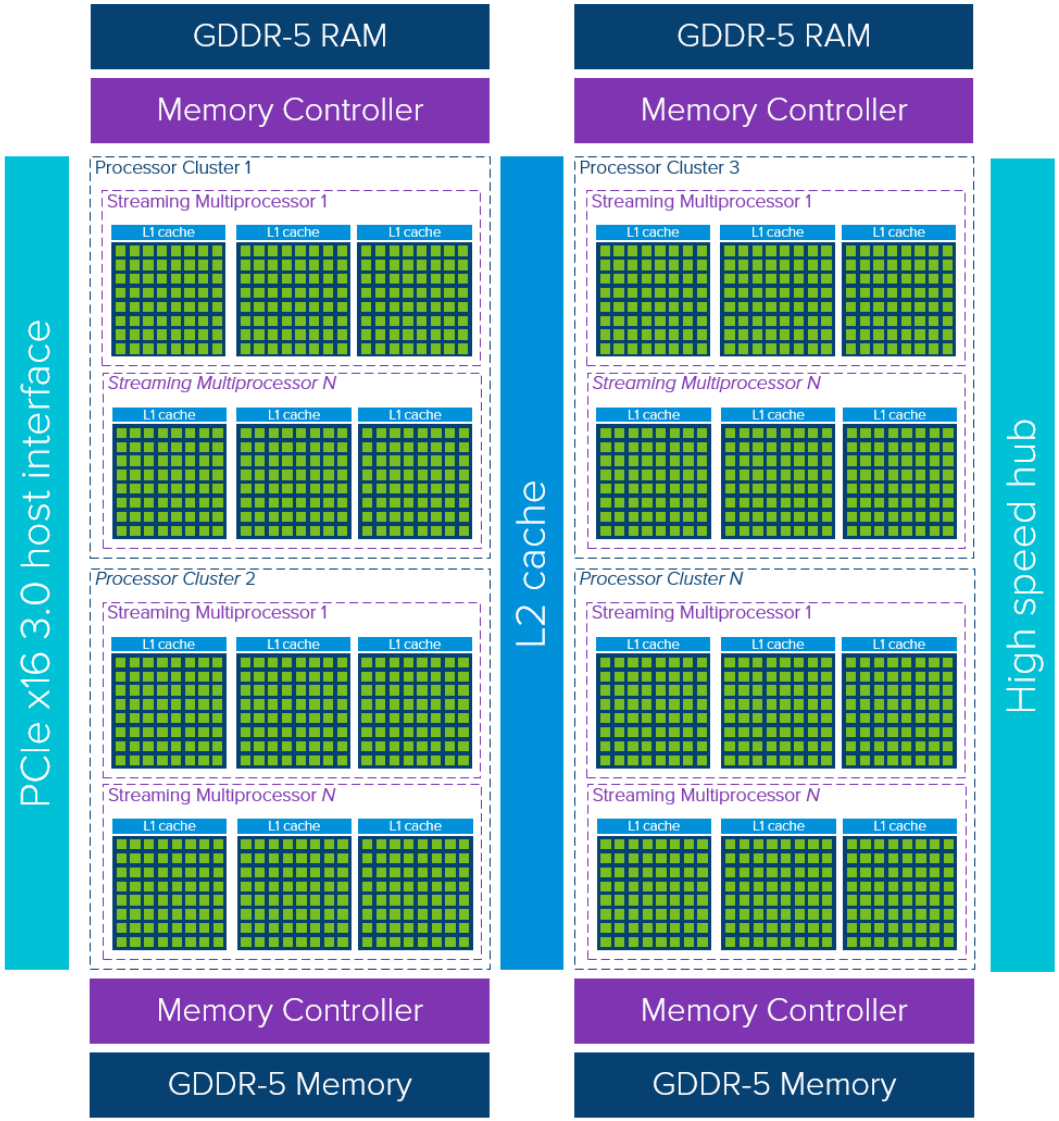
Many of the costs are mitigated by modern languages providing thread-safe, optimized libraries.

Parallel paradigms: CPU



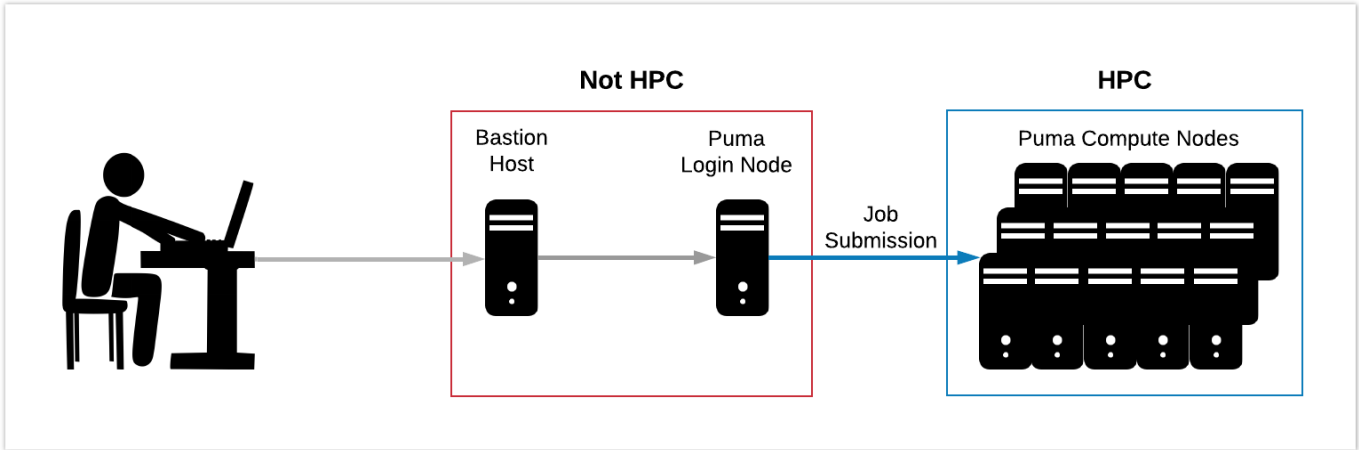
https://core.vmware.com/resource/exploring-gpu-architecture

Parallel paradigms: GPU



https://core.vmware.com/resource/exploring-gpu-architecture

Parallel paradigms: Multi-node



https://public.confluence.arizona.edu/display/UAHPC/Puma+Quick+Start

Word of warning

- Not all code is parallelizable! Improper implementations can lead to
  - Race conditions
  - Deadlock
  - Memory Corruption

badParallel.jl

```
a = 0;
for i in 1:1000
    global a += 1;
end
println("serial a = $(a)");

a = 0;
Threads.@threads for i in 1:1000
    global a += 1;
end
println("parallel a = $(a)");

# serial a = 1000
# parallel a = 881
```

CPU Thread parallelism

Julia: `helloWorld.jl`

```
numThreads = Threads.nthreads();
Threads.@threads for i in 1:numThreads
    println("Hello World!"*
"This is thread # $(Threads.threadid())");
end

# Hello World! This is thread # 1
# Hello World! This is thread # 6
# Hello World! This is thread # 3
# Hello World! This is thread # 4
# Hello World! This is thread # 5
# Hello World! This is thread # 2
```

Python: `helloWorld.py`

```
import threading;
from time import sleep;
import numpy as np;

def helloWorld():
    sleep(np.random.random());
    print("Hello world! This is {}"
          .format(threading.current_thre

if __name__ == "__main__":
    for i in range(6):
        t = threading.Thread(
            target=helloWorld,args=[]);
        t.start()

# Hello world! This is Thread-4
# Hello world! This is Thread-1
# Hello world! This is Thread-6
# Hello world! This is Thread-2
# Hello world! This is Thread-5
# Hello world! This is Thread-3
```

GPU Thread parallelism

- Even easier introduction to Cuda

Node parallelism via Slurm

References & Further Reading

- https://core.vmware.com/resource/exploring-gpu-architecture
- https://public.confluence.arizona.edu/display/UAHPC/Puma+Quick+Start
- https://www.tensorflow.org/guide/gpu
- https://developer.nvidia.com/blog/even-easier-introduction-cuda/